In many situations, embedded *menus represent an attractive alternative to* the more traditional *explicit* menus, particularly in *touchtext, spelling checkers, language-based program editors, and graphics-based systems.*

EMBEDDED MENUS: SELECTING ITEMS IN CONTEXT

LARRY KOVED and BEN SHNEIDERMAN

When compared to command driven systems, computer menu systems are appealing because they reduce memorization of commands, reduce training, and structure the user's decision making [12]. Menus can be categorized as either *embedded* or *explicit* [6], the difference being the context in which the menu items are presented.

Explicit menus (Figure 1) usually supply an explicitly enumerated list of items from which the user selects by typing a number or letter; a variant to this theme highlights or capitalizes the first letter of the selectable item. The use of icons, where all selectable icons are displayed on the screen, is a kind of enumeration and therefore also a form of explicit menu. Instead of entering numbers, letters, or icons, some systems permit the user to point to an item in the menu by physically touching the screen (if a touch screen is used), or by using arrow keys or a mouse. The item to be selected is highlighted (e.g., intensified, underlined, or put in reverse video), and pressing another key or button selects the item.

While explicit enumeration of menu items poses certain obvious advantages over command driven systems, in some situations explicit menus can themselves be inefficient: Listing items wastes viewing space on the computer display, and extracting

information from the original context to construct menu items may mean the items have to be excessively verbose to be meaningful.

Embedded menus, where menu items are embedded within the information being displayed on the screen, in some respects represent an improvement on the more traditional explicit menu. In embedded menus (Figure 2), highlighted or underlined words or phrases *within* the text become the menu items, and are selectable using the commonly used touch screen, cursor, and mouse methods cited above.

Our own experience with embedded menus began in the interest of providing adequate man–machine interfaces for two textual databases: The Interactive Encyclopedia Systems (TIES), a European history database functioning in a museum environment [13], and the OnLine Maintenance Manual (OLMM) system, an on-line maintenance manual for electrical and mechanical equipment repair [7, 15]. In this article, we review the use of embedded menus in these two specific systems and examine the more general application of embedded menus in interactive spelling checkers, language-based program editors, and interactive graphics systems. In so doing, we address the relative advantages and disadvantages of embedded menus in different contexts, highlighting areas of equivocation where more research is warranted.

THE INTERACTIVE ENCYCLOPEDIA SYSTEM

TIES was designed for a museum environment where visitors walk up to a TIES machine and explore a database on European history [13]. Since most TIES users were expected to be novices (i.e., people who had never used TIES or even, perhaps, a computer), the simplicity of the user interface was considered paramount. To this end, instead of extracting menu items and displaying them as an explicit menu, selectable items were highlighted directly in the text (Figure 3), a method of displaying text with embedded items that has since become known as *touchtext*.

In TIES, there are three active keys—move cursor left, move cursor right, and select menu item—each of which is activated by a single key stroke. The initial screen presents an article with highlighted

```
EVENTS: ANSCHLUSS                    Page 2 of 7
The victorious Allies disapproved of such
a union and specifically forbade it in
both the Treaty of Versailles and the
Treaty of St. Germain-en-Laye. Austrian
nationalism remained weak throughout
the interwar period (1918-1939). During
these years, Austria, like Germany, gave
rise to a number of right-wing and
fascist political movements. Indeed,
Adolph Hitler's own Nazi Party had
a sizable Austrian branch. In 1934,
Engelbert Dollfuss, a member of the
Christian Social Party, destroyed the
First Republic's fragile parliamentary
democracy and established a right-wing
dictatorship.
─────────────────────────────────────────────
Next Page      Previous Page      Return to FREUD, SIGMUND
```

The above example is an embedded menu from a TIES database. The menu items are shown in bold type, and the cursor is an inverse video bar.

FIGURE 3. TIES Menu

```
Enter the number of the item of interest:
1 — New Books
2 — All Books
3 — New Periodicals
4 — All Periodicals
5 — Other Library Services

Enter the first letter of the word to make
a selection:

Compile Edit Run

Save Options Quit
```

The menu items are explicitly enumerated.

FIGURE 1. Explicit Menu

```
   The options for the NESW intercom system
are controlled by a set of switches on the
controller card. Locate the controller card
and remove it from the system unit.
Find the serial number.
   Set the switches based on the serial num
ber. Once the switches are set,
insert the controller card.
```

The menu items are the underlined phrases: To make a selection, the user moves the cursor to the desired phrase and presses the select key.

FIGURE 2. Embedded Menu

phrases—the selectable menu items—and the user positions the cursor on the phrase for which more detail is desired, and selects the item; a new article is retrieved, and the process is repeated, if desired.

The top line of the screen shows the article title and page number. At the bottom of the screen are additional page-turning commands listed in the form of an explicit menu: "Next Page," "Previous Page," and "Return to (previous article)." Menu items not applicable are omitted (e.g., the "Previous Page" option when the first page of an article is being displayed).

All in all, TIES provides a very simple strategy for accessing information. To the user, the database is a network of related articles that can be retrieved by starting with an introductory article and then making menu selections. At every node in the network (except the introduction), the user can request a return to a previous article.

ONLINE MAINTENANCE MANUAL

In many respects, the OLMM system is similar to TIES: It also uses touchtext, and the syntax and semantics of the two systems are almost the same [7, 15]. The OLMM is designed to be used with databases containing training, diagnostic, and repair

manuals: Its objective is to replace paper manuals with on-line versions through the use of alternate modes of presentation and database structuring.

Replacing paper manuals with on-line versions, however, is fraught with problems: First of all, the process of flipping through the pages of a book to find a section of interest is a familiar one to most people: Place markers are commonly inserted, and even notations made in margins—possibilities not frequently available with on-line systems. Also, the informational content per page in a book is greater than on a 24-line screen. Furthermore, in most cases, manuals are stored on-line as they would be printed on paper, and typically suffer from overly complex retrieval methods, limited display space, and poor screen readability [2, 4, 8, 16].

To deal with the diagrams and pictures commonly found in maintenance manuals, graphical data in the OLMM are associated with each database node so that, each time a new section of the manual is retrieved from the database, an illustration is also retrieved and displayed on an accompanying graphics screen. In the initial implementation of the OLMM, only simple line drawings were included, although more complex graphics are possible—even animation or videodiscs. The types of output possible are limited only by the capabilities of the graphics hardware and software, and the writer's imagination.

SPELLING CHECKING AND CORRECTING PROGRAMS

Touchtext is only one example of an embedded menu. Another familiar example is the use of embedded menus in interactive spelling programs, where the document is displayed to the user as it was given to the spelling program, but words detected as possibly misspelled are highlighted or underlined; the user points to a highlighted misspelled word and requests that the spelling program display an explicit menu of the possible correct spellings (Figure 4). Once the user selects a correct spelling from the menu, the spelling program substitutes the correct for the incorrect spelling. If the user determines that a highlighted word is in fact spelled correctly, he or she can simply skip past the word, avoiding an unnecessary search in the dictionary. Typically a command, possibly a single keystroke, moves the cursor to the next misspelled word in the document.

The style checker—a variant of the spelling program—goes one step further and detects possible misuses of a word in a given context, highlighting the possibly incorrect word. In a document containing the word "than" when the correct word should probably be "then," the program will highlight the word "than" to indicate that it may be incorrectly used.
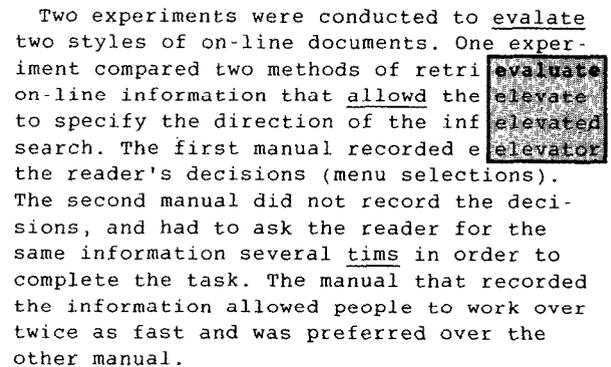
The misspelled words are highlighted in the embedded menu. Since the word "evaluate" is considered to be the most likely correct spelling, it is at the top of the list of words in the explicit menu.

FIGURE 4. Spelling Checker with Embedded Menu and Possible Correct Choices

The use of embedded menus in spelling programs is a natural one as the possibly misspelled words are presented within the original context of their usage. The alternative, taking words out of context, greatly increases the difficulty of determining whether a particular word is actually misspelled or simply not in the program's dictionary. If a spelling program were to extract all supposedly misspelled words from the document and simply display them as an explicit menu without showing where in the document the misspelled words were located [1], it would be difficult to decide if the word "Martian" was the adjective to describe an inhabitant of the planet Mars, or whether the word should be "Martial." In a situation like this, the user simply guesses whether or not the word is correct. Moreover, it is possible that a word may be correctly spelled in one place in the document and incorrectly in another.

LANGUAGE-BASED PROGRAM EDITORS

In recent years, there has been increased interest in language-based editors like the Cornell Program Synthesizer [14]. Language-based editors differ from text editors in that they incorporate the syntax of the programming language to help create syntactically correct programs by only permitting the entry of information that maintains a syntactically correct program. One beneficial by-product of these editors is the automatic pretty printing of programs.

In a language-based editor, the program is maintained as a syntax tree. The user creates a new program in a top-down fashion, generating program

constructs and filling in the details. In many re-
spects, it is akin to a fill-in-the-blanks style of edit-
ing. The user moves the cursor to the desired part of
the program, for example, the ⟨*statement*⟩ nonter-
minal in the derivation tree. The ⟨*if*⟩ command then
causes the editor to expand the ⟨*statement*⟩ nonter-
minal into a new subtree containing the derivation
subtree for an IF-THEN statement, with nontermi-
nals ⟨*boolean-expression*⟩ and ⟨*statement*⟩ (see Figure
5). Each of these nonterminals may be expanded or
filled in with appropriate terminal and nonterminal
symbols, or modified, or deleted.

Because the program is represented as a syntax
tree, it is possible to hide the details of a program
subtree—for example, the details of a procedure
body or a while loop (Figure 6). This procedure is
known as holophrasting [3]. In this way, the user is
able to manipulate the display of the subtrees so that
only certain portions of the syntax trees are dis-
played, and the details of other portions are hidden
from view. By pointing to the appropriate subtree,
the user is able to expand (zoom) or contract (un-
zoom) the view of these hidden subtrees.

Language-based editors also make it possible to
easily locate and display the declarations of an iden-
tifier. If a user points to a variable in a program, the
editor can search through the syntax tree, using the
language's scoping rules, to locate the declaration of

```
Program Example (Input, Output);

var
    ⟨identifier⟩ : ⟨type⟩;

begin
    ⟨STATEMENT⟩
end.

     (a)
```

```
Program Example (Input, Output);

var
    ⟨identifier⟩ : ⟨type⟩;

begin
    if ⟨boolean-expression⟩ then
        ⟨statement⟩;
end.

     (b)
```

Part (a) is a basic program template. By pointing to ⟨state-
ment⟩ and selecting the ⟨if⟩ function, the nonterminal was
expanded, resulting in part (b).

FIGURE 5. Language-Based Editor

```
Program Example (Input, Output);

var
    ⟨identifier⟩ : ⟨type⟩;

Procedure Hidden;
    . . .

begin
    while Condition1 do
    . . .

    while Condition2 do
    . . .

    ⟨statement⟩
end.
```

The body and declarations of the procedure *Hidden* are sup-
pressed, indicated by the ellipses; and the body of the pro-
gram contains two while loops, the bodies of which are also
suppressed. This suppression of details allows more of the
high-level structure of the program to be displayed on the
screen.

FIGURE 6. Holophrasting

the variable in the current context. The same is true
of procedure and function declarations.

An explicit menu system that would perform a
comparable function for language-based editors
would be extremely cumbersome or seem very un-
natural. With an explicit menu, the editor would ask
the user which subtree should be manipulated based
on its location within the program (see Figure 7, on
the next page), and by entering a number between 1
and 4, the user would designate the assignment
statement to be modified or deleted. The explicit
enumeration of menu items, however, would con-
sume a large portion of the available display space,
thereby reducing the amount available for program
text; it would also remove the statements from con-
text, making the user's decision more difficult as
several items may be syntactically identical.

FURTHER EXAMPLES

Spatial Data Management Systems (SDMS) [5] uses a
technique of displaying general database information
by using graphics. Presented with a map of the
United States, a user wishing to see all counties with
a population greater than 1,000,000 would be shown
those locations satisfying the criterion on the graph-
ics display. The user may then use a pointing device
to select a particular region of the map and thereby
retrieve a more detailed map from the database. By
entering more selection criteria (e.g., manufacturing
or industry), the user may continue the database
search process. The leaves of the database are repre-

```
Program Example (Input, Output);
        . . .
Function Compute (Arg : Integer);
begin
        . . .
    a:=a+1;
    b:=Compute(a);
    a:=a+b;
    c:=Compute(a);
    a:=a+b;
    b:=Compute(a);
    a:=a+b;

        . . .
end.
_____

Enter the number of the statement to be
deleted:

(1)  a:=a+1;
(2)  a:=a+b;
(3)  a:=a+b;
(4)  a:=a+b;
```

Since the assignment statements are removed from their context, it is more difficult to determine which of the four assignment statements should be deleted.

FIGURE 7. Using an Explicit Menu to Modify a Program

sented as icons displayed on the screen and may be selected by the user. At any time, the user can undo the effects of the selection process by returning to the less detailed maps.

Embedded menus are also found in graphics-based systems. For example, when designing a VLSI chip the user interacts with the system through direct manipulation of the graphical objects [9, 11]. The objects displayed on the screen form an embedded menu from which the user may make selections. A selection is made by pointing to a desired object and then requesting that it be moved, copied, deleted, etc.

DISCUSSION
Despite the many obvious benefits of embedded menus, more research is needed. In the case of touchtext, for example, it is not clear what kind of negative effects may arise from the use of high-lighted or underlined menu items. It is possible that the highlighting of phrases is disruptive, causing re-duced reading speed and comprehension. Also, since embedded menu items can be selected in any order, the novice or inexperienced user may get lost by jumping around in the material rather than travers-ing the database in an in-order or sequential fashion.

The touchtext systems do not enforce a search order. On the other hand, for experts, or searches initiated to locate very specific information, the ability to skip material or peruse the material in a different order may be enormously beneficial and may dramatically reduce search time [7].

Another possible drawback, particularly for touch-text systems, is that the mixing of information with menu selection items may be disruptive to the learn-ing process in the sense that the user may be in-clined to examine a particular subject or subjects in detail without first getting an appreciation of the overall context. By traversing down through several levels of the database, the user may forget the origi-nal context in which the material was retrieved.

For the frequent or sophisticated user, embedded menus that require frequent traversals of familiar paths to access details of the system may become cumbersome: An alternative may be offering short-cuts or command languages as a means of bypassing certain menus.

Finally, more research is needed in the area of computerized books and documents, which are not well understood from the cognitive point of view and for which improved man–machine interfaces are necessary. We do know that reading material from most currently used computer screens is slower than from paper, although the specific reasons are not completely understood [2]. To compensate, we can try to use the technology and capabilities of the computer to provide new means of storing, locating, retrieving, and displaying information. This new technology, however, presents the users with an en-vironment that is quite different and not particularly well understood: the syntax and semantics of com-puterized books and documents being quite distinct from that of their paper counterparts. Special train-ing will be needed, although the use of embedded menus may help reduce the amount of time needed.

EXPERIMENTAL RESULTS
Several experiments conducted with embedded menus seem to indicate that, all things considered, embedded menus may represent an attractive alter-native to traditional menus and command syntax.

One of the first experiments performed to com-pare embedded and explicit menus used TIES with a database describing the Student Union of the Uni-versity of Maryland at College Park [10]. A within-subject experimental design required that subjects search the TIES database for answers to 20 questions about the Student Union, all in a 15-minute period. The number of correctly answered questions was recorded for each experimental condition—embed-ded versus explicit menu. The results showed that

many more questions were answered correctly using embedded menus than with explicit menus ($p <$ 0.001); in addition, fewer screens were viewed when using embedded menus ($p < 0.001$), and the subjects actually preferred the embedded over the explicit menu ($p < 0.001$).

A second between-subjects experiment was conducted with the OLMM system to compare embedded menus versus page-turning commands for on-line manuals [7, 15]. In the page-turning mode, the text on the screens was augmented with the page number for each embedded menu item: However, instead of allowing embedded menu selection, page-turning mechanisms were provided—forward and backward and "first-page" keys, and page-number entry for direct access to specific pages. Results gathered from a posttest questionnaire revealed that embedded menus were preferred to the page-turning method ($p < 0.03$) even though the design of the material in the embedded menu condition prevented subjects from solving problems as fast as with the page-turning technique ($p < 0.01$). This slowness was due to the fact that the design forced subjects to view more pages to solve each problem ($p < 0.04$), which suggests a need to investigate alternative rapid access strategies, particularly for frequent users.

A third experiment involving the OLMM system was conducted to study the performance of people using a novel textual database searching technique known as pruning [7]. In this within-subject experiment using embedded menus, a pruning technique was used to trim text not relevant to the task at hand. This reduced both the amount of text that had to be read, and the complexity of the questions that had to be answered to complete each problem. Using embedded menus and the pruning technique, problems were solved in less than half the time ($p <$ 0.001) and required the viewing of fewer pages ($p <$ 0.001); in addition, less time was spent viewing each page ($p < 0.001$). These results are considered important because, for many applications, reading from manuals printed on paper is faster than from computer displays [2, 4, 8, 16]. The results also suggest that embedded menus, together with the pruning technique, may make on-line manuals an effective alternative to printed manuals.

CONCLUSION
Using embedded menus makes it easier to avoid computer-related syntax and semantics issues when referring directly to the object being manipulated. The embedded menu can be much simpler than a comparable explicit menu since only a simple cursor movement is required to find an object of interest: In

the case of textual information, the cursor control may be as simple as a single cursor movement key; for graphics-based systems, the cursor may be cross hairs pointing to the object of interest. Of course, the newer pointing devices—the mouse, track ball, and touch screen—can also be used to point to either textual or graphical information. The selection mechanism in this case is either a single keystroke or the click of a mouse button.

In an explicit menu approach, the viewing becomes divorced from the selection process, and may become unwieldy if the selection command must include operation type as well as operands to specify the target of the operation. When this happens, the association between displayed information and menu items may become less clear.

One of the most appealing aspects of embedded menus is the direct manipulation approach to controlling the application. It allows the user to point directly to an object of interest, with the underlying system performing the desired operation. Embedded menus in some respects resemble WYSIWYG (what-you-see-is-what-you-get) text editors. There is no arbitrary syntax associated with the menu selection process: Instead, the syntax consists of simply moving the cursor, or rectangular window in graphics systems, to a desired location on the display, and requesting that an operation be performed on the currently referenced object.

With the emergence of very small portable computers and a space limitation of 24 or 25 lines by 80 columns for many of today's screens, space saving techniques will remain important. Although in the future we expect larger computer displays to become more common and the conserving of screen space less critical for some applications, we believe that embedded menus will remain an important human–computer interaction technique.

REFERENCES
1. Bentley, J. Programming pearls: A spelling checker. *Commun. ACM* 28, 5 (May 1985), 456–462. Techniques for implementing spelling checkers.
2. Gould, J.D., and Grischkowsky, N. Doing the same work with hard copy and with cathode-ray tube (CRT) computer terminals. *Hum. Factors 26*, 3 (June 1984), 323–337. A study was conducted to determine the potential negative effects CRT displays may have on people using them. One result of the study showed that people read from computer displays slower than from paper.
3. Hansen, W.J. User engineering principles for interactive systems. In *Proceedings of the Fall Joint Computer Conference*, vol. 39 (Las Vegas, Nev., Nov. 16–18). AFIPS Press, Montvale, N.J., 1971, pp. 523–532. The Emily program editor is described, and also gives guidelines for the design of user interfaces.
4. Hansen, W.J., Doring, R., and Whitlock, L.R., Why an examination was slower on-line than on paper. *Int. J. Man-Mach. Stud. 10*, 5 (Sept. 1978), 507–519. Students were given an examination on-line, and the reasons for why the examination took longer to use were studied.
5. Herot, C.F. Graphical user interfaces. In *Human Factors and Interactive Computer Systems*, Y. Vassiliou, Ed. Ablex, Norwood, N.J., 1984, pp. 83–103. A description of the Spatial Data Management System (SDMS).

6. Koved, L. *Implicit versus Explicit Menus*. IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y., 1984. Embedded and explicit menus are described, and some examples are given.
7. Koved, L. Restructuring textual information for online retrieval. Master's thesis, TR-1529, Dept. of Computer Science, Univ. of Maryland, College Park, July 1985. (Also, IBM Res. Div. Rep. RC 11278, IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y.) Two different techniques for storing manuals on-line are used in experiments to evaluate their design. The "pruning" technique enabled simple tasks to be accomplished in half the time than if the technique was not used.
8. Mills, C.B., and Weldon, L.J. Reading from computer screens. Tech. Rep. CAR-TR-94, Human–Computer Interaction Laboratory, Center for Automation Research, Univ. of Maryland, College Park, Oct. 1984. Survey of issues relating to the readability of computer screens.
9. Newcomer, J.M., Ed. *SUDS Users' Manual*. Carnegie-Mellon Univ., Pittsburgh, Pa., 1980. Describes a logic and PC board design system.
10. Powell, D. *Experimental Evaluation of Two Menu Designs for Information Retrieval*. Dept. of Computer Science, Univ. of Maryland, College Park, 1985. An experiment was conducted to compare embedded and explicit menus.
11. Shneiderman, B. Direct manipulation: A step beyond programming languages. *Computer 16*, 8 (Aug. 1983), 57–69. Describes direct manipulation, and gives examples.
12. Shneiderman, B. Designing menu selection systems. *J. Am. Soc. Inf. Sci*. To be published.
13. Shneiderman, B., and Ostroff, D. TIES authoring system. Univ. of Maryland, College Park, Dec. 1985. Author's manual for creating TIES databases.
14. Teitelbaum, T., and Reps, T. The Cornell Program Synthesizer: A syntax-directed programming environment. *Commun. ACM 24*, 9 (Sept. 1981), 563–573. Describes a language-based (syntax-directed) program editor.
15. Weldon, L.J., Mills, C.B., Koved, L., and Shneiderman, B. The structure of information in online and paper technical manuals. In *Proceedings of the Human Factors Society Conference* (Baltimore, Md., Sept. 29–Oct. 3). Human Factors Society, Santa Monica, Calif., 1985, pp. 1110–1113. An experiment on user performance, comparing paper and on-line manuals.
16. Wright, P., and Lickorish, A. Proof-reading texts on screen and paper. *Behav. Inf. Technol. 2*, 3 (July–Sept. 1983), 227–235. Both speed and accuracy are poorer when using a CRT screen than if the same task were done on paper.
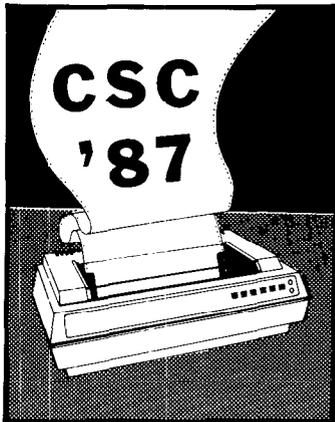
Authors' Present Addresses: Larry Koved, IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598; Ben Shneiderman, Dept. of Computer Science and Human–Computer Interaction Laboratory, University of Maryland, College Park, MD 20742.